CS370

Symbolic Programming
Declarative Programming

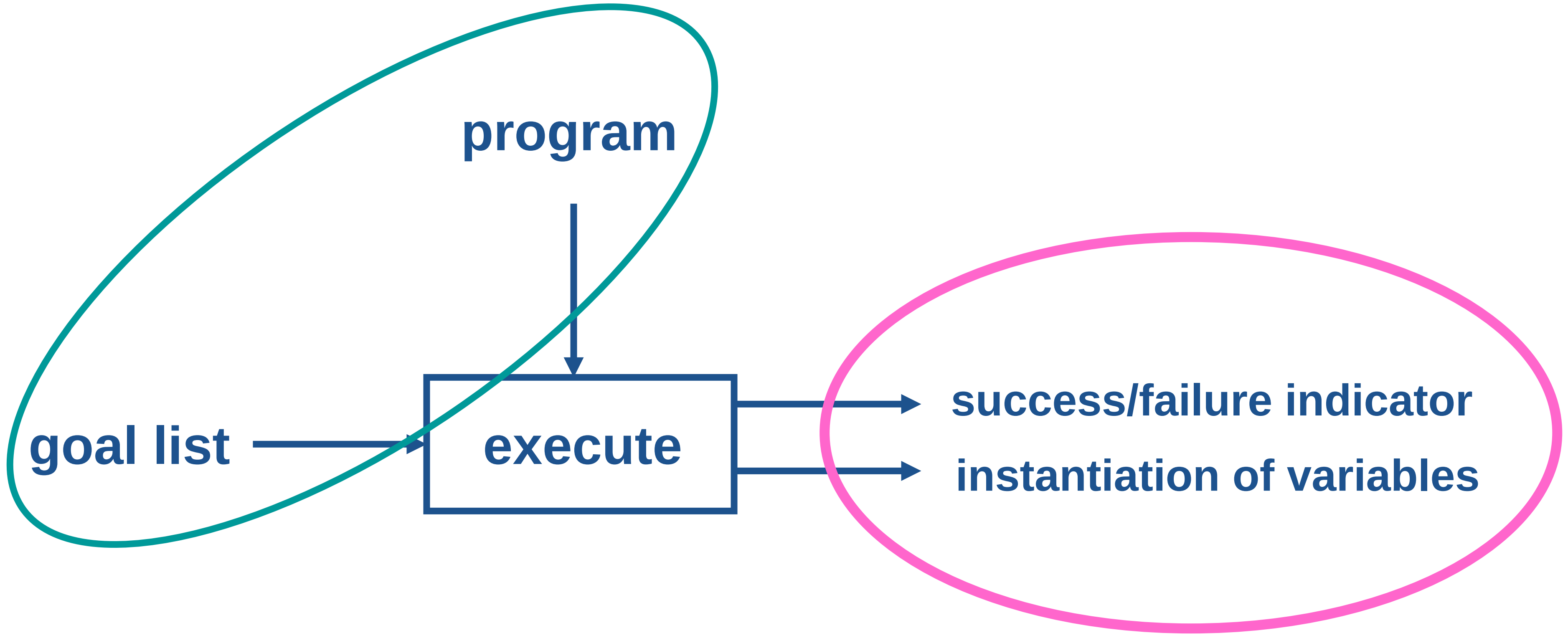
LECTURE 4: Syntax and Meaning of Prolog Programs

Jong C. Park
park@cs.kaist.ac.kr
Computer Science Department
Korea Advanced Institute of Science and Technology
http://nlp.kaist.ac.kr/~cs370

Syntax and Meaning of Prolog Programs

- Data objects
- Matching
- Declarative meaning of Prolog programs
- **Procedural meaning**
- **Example: monkey and banana**
- **Order of clauses and goals**
- **The relation between Prolog and logic**

Procedural Meaning



Procedural Meaning

- **To execute a list of goals, G_1, G_2, ..., G_m, the procedure execute does the following:**
  - If the goal list is empty then terminate with success.
  - If the goal list is not empty then continue with (the following) operation called ‘SCANNING’.

Procedural Meaning

- **SCANNING**:
  - Scan through the clauses in the program from top to bottom until the first clause, C, is found such that the head of C matches the first goal G1.
  - If there is no such clause then terminate with failure.
  - Suppose that C is of the form H :- B1, ..., Bn. Rename the variables in C to obtain a variant C' of C, such that C' and the list G1, ..., Gm have no common variables. Let C' be H' :- B1', ..., Bn'.
  - Match G1 and H'; let the resulting variable instantiation be S.
  - In the goal list G1, G2, ..., Gm, replace G1 with the list B1', ..., Bn', obtaining a new goal list B1', ..., Bn', G2, ..., Gm .
  - Substitute the variables in this new goal list with new values as specified in the instantiation S, obtaining another goal list B1'', ..., Bn'', G2', ..., Gm' .

Procedural Meaning

- Execute this new goal list.
  - If the execution of this new goal list terminates with success then terminate the execution of the original goal list also with success.
  - If the execution of the new goal list is not successful then abandon this new goal list and go back to SCANNING through the program.
  - Continue the scanning with the clause that immediately follows the clause C and try to find a successful termination using some other clause.

Procedural Meaning

PROGRAM

```
big(bear).
big(elephant).
small(cat).
brown(bear).
black(cat).
gray(elephant).
dark(Z) :- black(Z).
dark(Z) :- brown(Z).
```

QUESTION

```
?- dark(X), big(X).
```

EXECUTION TRACE

- dark(X),big(X).
- black(X),big(X)
- big(cat)
- No clause for big(cat).
- Backtract to: black(X),big(X)
- No clause for black(X).
- Backtract to: dark(X),big(X).
- brown(X),big(X)
- brown(bear)
- big(bear)
- X=bear

Example: monkey and bana

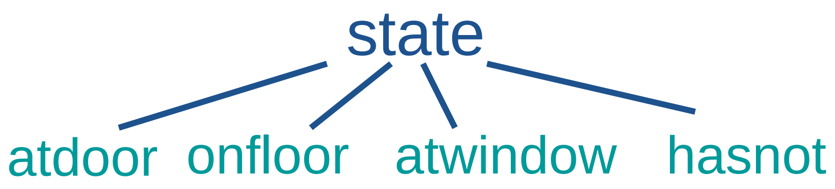
⊙ Problem Description

- Situation
  - There is a monkey at the door into a room.
  - In the middle of the room a banana is hanging from the ceiling.
  - The monkey is hungry and wants to get the banana, but he cannot stretch high enough from the floor.
  - At the window of the room there is a box the monkey may use.
- The monkey can perform the following actions:
  - walk on the floor, climb the box, push the box around (if it is already at the box) and grasp the banana if standing on the box directly under the banana.
- Can the monkey get the banana?

Example: monkey and banana

⊙Representation

- The monkey world is always in some state that can change in time.
- The current state is determined by the positions of the objects.
- The initial state of the world is determined by:
  - Monkey is at door.
  - Monkey is on floor.
  - Box is at window.
  - Monkey does not have banana.



state(_,_,_,has)

Example: monkey and banana

- **Allowed moves**
  - Grasp banana
  - Climb box
  - Push box
  - Walk around
- **Not all moves are possible in every possible state of the world.**
  - move(State1,Move,State2)

```
move(state(middle,onbox,middle,hasnot),
         grasp,
         state(middle,onbox,middle,has) ).
```

Example: monkey and banana

- **The monkey on the floor can walk from any horizontal position Pos1 to any position Pos2.**
  - The monkey can do this regardless of the position of the box and whether it has the banana or not.

    ```
    move(state(Pos1,onfloor,Box,Has),
            walk(Pos1,Pos2),
            state(Pos2,onfloor,Box,Has) ).
    ```

    - The move executed was ‘walk from Pos1 to Pos2’.
    - The monkey is on the floor before and after the move.
    - The box is at some point Box which remained the same after the move.
    - The ‘has banana’ status Has remains the same after the move.

Example: monkey and banana

- **Question**
  - Can the monkey in some initial state State get the banana?

```
?- canget(State).

canget(state(_,_,_,has) ).
canget(State1) :-
        move(State1,Move,State2),
        canget(State2).
```

Example: monkey and banana

- **Program**

```
move(state(middle,onbox,middle,hasnot),
        grasp, state(middle,onbox,middle,has) ).
move(state(P,onfloor,P,H),
        climb, state(P,onbox,P,H) ).
move(state(P1,onfloor,P1,H),
        push(P1,P2), state(P2,onfloor,P2,H) ).
move(state(P1,onfloor,B,H),
        walk(P1,P2), state(P2,onfloor,B,H) ).
canget(state(_,_,_,has) ).
canget(State1) :- move(State1,Move,State2),
        canget(State2).
```

Order of clauses and goals

- **Danger of infinite looping**

```
p :- p.
?- p.
```

- **Program variations**
  - through reordering of clauses and goals

```
predecessor(Parent,Child) :-
     parent(Parent,Child).
predecessor(Predecessor,Successor) :-
    parent(Predecessor,Child),
    predecessor(Child,Successor).
```

Order of clauses and goals

- Four variations of the predecessor program

```
pred1(X,Z) :- parent(X,Z).
pred1(X,Z) :- parent(X,Y),
              pred1(Y,Z).
```

```
pred2(X,Z) :- parent(X,Y),
              pred2(Y,Z).
pred2(X,Z) :- parent(X,Z).
```

```
pred3(X,Z) :- parent(X,Z).
pred3(X,Z) :- pred3(X,Y),
              parent(Y,Z).
```

```
pred4(X,Z) :- pred4(X,Y),
              parent(Y,Z).
pred4(X,Z) :- parent(X,Z).
```

The relation between Prolog and logic

- **Prolog**
  - As defined in relation to mathematical logic
    - First-order predicate logic
    - Clause form
    - Horn clauses
    - Resolution theorem proving
    - Unification in logic
  - As a programming tool

Summary

- **The procedural meaning does depend on the order of goals and clauses.**
  - The order can affect the efficiency of the program.
  - An unsuitable order may even lead to infinite recursive calls.
- **Given a declaratively correct program, changing the order of clauses and goals can improve the program’s efficiency while retaining its declarative correctness.**
  - Reordering is one method of preventing indefinite looping.